

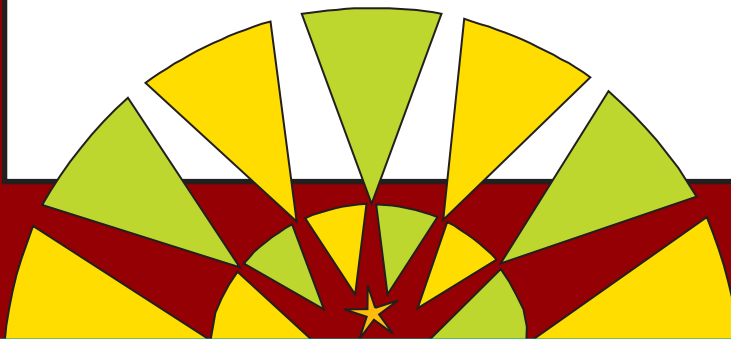
CHAPTER TWO

USING VARIABLES

Now that we have discussed some PHP background information and learned how to create and publish basic PHP scripts, let's explore how to use PHP variables. We will discuss how to define variable names, set variable values, and use some basic variable operators. Since PHP variables can receive input values from HTML forms, we'll also review how to create and use HTML form elements (such as checklists, radio boxes, text boxes, and submit buttons) and then see how to use them to pass data to PHP scripts.

Chapter Objectives

- ★ To learn how to store and access data in PHP variables
- ★ To understand how to create and manipulate numeric and string variables
- ★ To review how to create HTML input forms
- ★ To learn how to pass data from HTML forms to PHP scripts



Using PHP Variables

Variables are used to store and access data in computer memory. A **variable name** is a label used within a script to refer to the data. You can assign a data value to the variable name, change that value, print it out, and perform many different operations (for example, multiplication, addition). In PHP, variables can hold numeric data (for example, 1, 1239.12, -123), character string data (for example, “Extra Large”, “John Smith”, “Rhode Island”), or even entire *lists* of numeric or string data called arrays. Let’s concentrate on numeric and character string variables now; we will cover arrays in Chapter Five.

In your PHP scripts, you can create and use your own variables by assigning a value to a variable name. Simply place the variable’s name on the left side of an equal sign (=) and the value on the right side of the equal sign. That value will then be stored or saved into computer memory. The PHP statements shown in Figure 2.1 use two variables: `$cost` and `$months`.

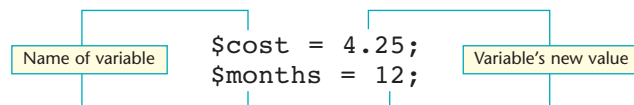


Figure 2.1 Assigning Values to Variables

Note that each PHP statement ends with a semicolon (;). Also note that the first statement assigns the value 4.25 to the variable `$cost` and the second statement assigns the value 12 to the variable `$months`. These assignments represent the current values of `$cost` and `$months`. You can also assign new values to variables. For example, consider the following PHP statements.

```
$days = 3;
$newdays = 100;
$days = $newdays;
```

This example uses two variables (`$days` and `$newdays`). At the end of these three lines, `$days` and `$newdays` both have values of 100.

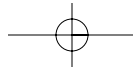
You can select just about any set of characters for a variable name in PHP, as long as you adhere to the following rules.

- ★ The first character must be a dollar sign (\$).
- ★ The second character must be a letter or an underscore character (_).

★ TIP Choose Descriptive Variable Names

It is good programming style to select variable names that help describe their function. For example, `$counter` is more descriptive than `$c` or `$ctr`, and `$dollars` is more descriptive than `$dol` or `$dlr`. The idea is to write self-documenting statements that other people will easily understand.

Thus `$baseball`, `$_time`, `$X`, `$Numb_of_bricks`, `$num_houses`, and `$counter1` are all valid PHP variable names. Conversely, `$123go`, `$1counter`, and `counter` are not valid choices.



★ **WARNING** Variable Names Are Case Sensitive

PHP variable names are case sensitive, so be careful about how you use upper- and lowercase letters in variable names. For example, \$numTimes, \$Numtimes, and \$NumTimes are all different variable names.

Combining Variables and the print Statement

In the first chapter, we used a `print` statement to output HTML tags and text. You can also use a `print` statement to output a variable's value by placing the variable name inside the double quotes of the `print` statement. That is, to print out the value of `$x`, write the following PHP statement:

```
print ("$x");
```

You can also include text to output using the `print` statement. The following code will output "Bryant is 6 years old".

```
$age=6;
print ("Bryant is $age years old.");
```

Consider the following sample PHP script and its browser output in Figure 2.2, which shows how to initialize variables (lines 5–6) and output their values (line 10). (Do not enter the numbers at the start of each line.)

```
1. <html>
2. <head> <title>Variable Example </title> </head>
3. <body>
4. <?php
5.     $first_num = 12;
6.     $second_num = 356;
7.     $temp = $first_num;
8.     $first_num = $second_num;
9.     $second_num = $temp;
10.    print ("first_num= $first_num <br>
        second_num=$second_num");
11. ?> </body> </html>
```

\$temp is assigned 12.
 \$first_num is assigned 356.
 \$second_num is assigned 12.

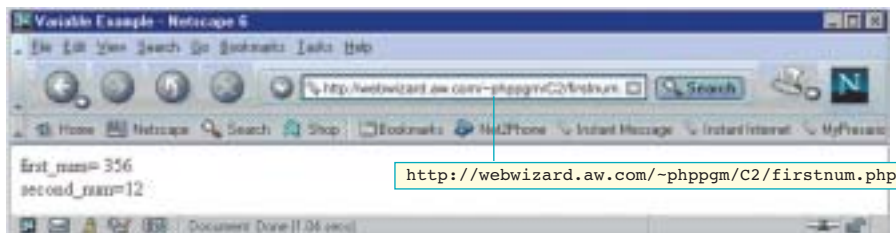
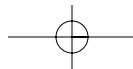
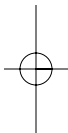
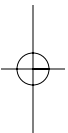


Figure 2.2 Output of a Script That Switches the Values of Two Variables



Here's a brief explanation of the script.

- ★ Lines 1–3 start the HTML document.
- ★ Lines 4–6 begin the PHP script, assigning the value 12 to `$first_num` and the value 356 to `$second_num`.
- ★ Line 7 assigns the value 12 to the variable `$temp`.
- ★ Lines 8–9 assign the values 356 to `$first_num` and then 12 to `$second_num`.
- ★ Line 10 outputs the values of `$first_num` and `$second_num`.
- ★ Line 11 first ends the PHP script and then the HTML document.

Using Arithmetic Operators

Using arithmetic operators, you can create expressions to manipulate variable data values. To do so, use **operators** such as a plus sign (+) for addition and a minus sign (-) for subtraction. For example, consider the following PHP statements.

```
<?php
    $apples = 12;
    $oranges = 14;
    $total_fruit = $apples + $oranges;
    print ("The total number of fruit is $total_fruit");
?>
```

These PHP statements would output “The total number of fruit is 26.”

Addition and subtraction are just two numeric operations that PHP supports. Table 2.1 lists the PHP common numeric operators and gives an example of each operator's use.

Table 2.1 Common PHP Numeric Operators

Operator	Effect	Example	Result
+	Addition	<code>\$x = 2 + 2;</code>	<code>\$x</code> is assigned 4.
-	Subtraction	<code>\$y = 3;</code> <code>\$y = \$y - 1;</code>	<code>\$y</code> is assigned 2.
/	Division	<code>\$y = 14 / 2;</code>	<code>\$y</code> is assigned 7.
*	Multiplication	<code>\$z = 4;</code> <code>\$y = \$z * 4;</code>	<code>\$y</code> is assigned 16.
%	Remainder	<code>\$y = 14 % 3;</code>	<code>\$y</code> is assigned 2.

The following example illustrates an HTML document with a PHP script that uses numeric operators. It calculates the total possible revenue and profit for a hypothetical theater assuming it has 12 rows with 20 seats in each row. The the-

ater also charges \$3.75 per ticket and has a fixed building cost of \$300. Figure 2.3 shows the script's output.

```

1. <html>
2. <head> <title>Variable Example </title> </head>
3. <body>
4. <?php
5.     $columns = 20;
6.     $rows = 12;
7.     $total_seats = $rows * $columns;
8.
9.     $ticket_cost = 3.75;
10.    $total_revenue = $total_seats * $ticket_cost;
11.
12.    $building_cost = 300;
13.    $profit = $total_revenue - $building_cost;
14.
15.    print ("Total Seats are $total_seats <br>");
16.    print ("Total Revenue is $total_revenue <br>");
17.    print ("Total Profit is $profit");
18. ?> </body> </html>

```

Assign 240 to \$total_seats.

Assign 900 (240 * 3.75) to \$total_revenue.

Assign 600 to \$profit (900-300).



Figure 2.3 Output of a Script Using Numeric Operators

Let's briefly examine this script.

- ★ Lines 1–3 start the HTML document.
- ★ Lines 4–7 start the PHP script, assigning the values 20 to `$columns`, 12 to `$rows`, and then 240 to `$total_seats`.
- ★ Lines 9–10 assign the values 3.75 to `$ticket_cost` and 900 to `$total_revenue` ($240 * 3.75$).
- ★ Lines 12–13 assign the values 300 to `$building_cost` and 600 to `$profit` ($900 - 300$).
- ★ Lines 15–17 output the values of `$total_seats`, `$total_revenue`, and `$profit`.
- ★ Line 18 first ends the PHP script and then the HTML document.

★ WARNING Using Variables with Undefined Values

PHP will not generate an error if you accidentally use a variable that does not have a value assigned to it. Instead, the variable will have no value (called a **null value**). When a variable with a null value is used in an expression PHP, PHP may *not* generate an error and may complete the expression evaluation. For example, the following PHP script will not generate an error when viewed over the Internet with a browser.

```
<?php
  $y = 3;
  $y=$y + $x + 1;           // $x has a null value
  print ("x=$x y=$y");
?>
```

The above script would output "x= y=4".

★ SHORTCUT Automatic Increment/Decrement Operators

Since it is common for scripts to increment or decrement a variable's value by 1 (for example, to decrement an inventory count), PHP supports two additional operators that automatically do this. If you precede a variable with ++ or -- PHP will automatically add or subtract 1 to the variable. For example, the following script would output "x=1 and now y=1".

```
<?php
  $x=0; $y=2;
  print "x=";
  print ++$x;
  --$y;
  print " and now y=$y";
?>
```

Writing Complex Expressions

You can include multiple operators in a statement to create complex expressions. When using multiple operators, you need to be aware of the **operator precedence rules** that define the order in which the operators are evaluated. For example, consider the following expression:

```
$x = 5 + 2 * 6;
```

The value of `$x` is either 42 or 17, depending on whether you evaluate the addition or the multiplication first. Because PHP evaluates multiplication operations before addition operations, this expression evaluates to 17. PHP follows the precedence rules listed below.

- ★ First it evaluates operators within parentheses.
- ★ Next it evaluates multiplication and division operators.
- ★ Finally it evaluates addition and subtraction operators.

To see how these precedence rules work, consider the PHP statements shown below. The first two equations are equivalent, and both evaluate to 80. The first statement uses the precedence rules given above, and the second clarifies these rules by using parentheses. The third statement uses parentheses to set a different order for operator evaluation, and it evaluates to 180.

```
$x = 100 - 10 * 2;
$y = 100 - (10 * 2);
$z = (100 - 10) * 2;
```

★ **TIP** Use Parentheses to Clarify Operator Precedence

Using parentheses to specify and clarify operator precedence is generally a good idea. It clarifies your intentions, makes your scripts easier to understand, and reduces errors.

As another example of using mathematical expressions, consider the following HTML document with a PHP script that calculates the average of three numbers. Figure 2.4 shows its output.

```

1. <html>
2. <head> <title>Expression Example </title> </head>
3. <body>
4. <?php
5.     $grade1 = 50;
6.     $grade2 = 100;
7.     $grade3 = 75;
8.     $average = ($grade1 + $grade2 + $grade3) / 3;
9.     print ("The average is $average");
10. ?> </body> </html>

```

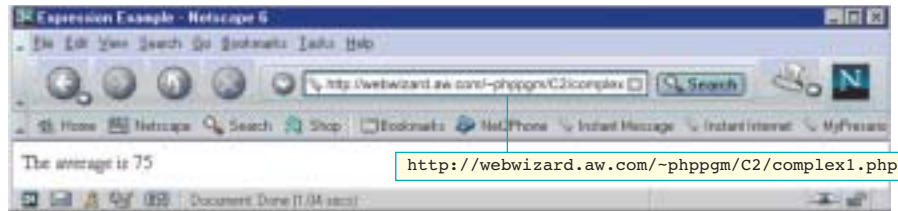


Figure 2.4 Output of a Script That Calculates an Average

The following explanation covers the key lines from the script.

- ★ Lines 5–7 set the initial values of the variables `$grade1`, `$grade2`, and `$grade3`.
- ★ Line 8 calculates the average of the three variables, and then line 9 outputs the result.

Working with PHP String Variables

So far, we have discussed variables with numeric values. As previously mentioned, variables can also be assigned character string values. Character strings are used in scripts to hold data such as customer names, addresses, product names, and descriptions. Consider the following example.

```

$name="Christopher";
$preference="Milk Shake";

```

★ **WARNING** Be Careful Not to Mix Variable Types

Be careful when using expressions not to mix string and numeric variable types. For example, you might expect the following statements to generate an error message, but they will not. Instead, they will output "y=1".

```
<?php
  $x = "banana";
  $sum = 1 + $x;
  print ("y=$sum");
?>
```

Here the variable `$name` is assigned the character string "Christopher" and the variable `$preference` is assigned "Milk Shake".

Unlike numeric variables, you cannot add, subtract, divide, or multiply string variables. PHP uses other methods to manipulate string variables. Two ways to manipulate strings are with string operators and with string functions. Let's examine the concatenate operator and then look at some common string functions.

Using the Concatenate Operator

Use the concatenate operator when you want to combine two separate string variables into one. The concatenate operator is specified as follows.

```
$fullname = $firstname . $lastname;
```

Here the variable `$fullname` will receive the string values of `$firstname` and `$lastname` connected together. For example, consider the following partial PHP script.

```
$firstname = "John";
$lastname = "Smith";
$fullname = $firstname . $lastname;
print ("Fullname=$fullname");
```

This script segment would output "Fullname=JohnSmith".

★ **TIP** An Easier Way to Concatenate Strings

You can also use double quotation marks to create concatenation directly, as in the following example.

```
$FullName2 = "$FirstName $LastName";
```

This statement has the same effect as

```
$FullName2 = $FirstName . " " . $LastName;
```

Using PHP String Functions

There are several different PHP applications that may require the use of string manipulation functions. Functions work much like operators, and you can use PHP string functions to do things like determine the length of strings, get subsets of strings, and remove lead-

ing space characters. One use of string manipulation functions might be to validate or alter input received from HTML forms. (Receiving data from HTML forms is described later in this chapter.) For example, you may need to determine the number of characters entered into an HTML form field or remove extra spaces entered.

The `strlen()` Function

Most string functions require you to send them one or more arguments. Arguments are input values that functions use in the processing they do. Often functions return a value to the script based on the input arguments. For example, consider the `strlen()` function (Figure 2.5), which accepts an argument as a string variable and returns the number of characters in the string.

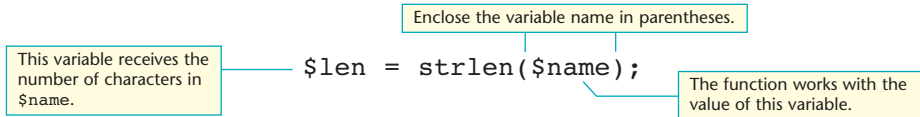
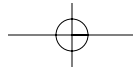


Figure 2.5 General Syntax of the `strlen()` Function

As an example of using the `strlen()` function, consider the following PHP script.

```
<?php
    $comments = "Good Job";
    $len = strlen($comments);
    print ("Length=$len");
?>
```

This PHP script would output “Length=8”.

The following subsections discuss the `trim()`, `strtolower()`, `strtoupper()`, and `substr()` PHP string functions and give examples of their usage.

The `trim()` Function

This function removes any blank characters from the beginning and end of a string. For example, consider the following script.

```
<?php
    $in_name = "    Joe    Jackson    ";
    $name = trim($in_name);
    print ("name=$name$name");
?>
```

This PHP script would output “name=Joe JacksonJoe Jackson”.

The `strtolower()` and `strtoupper()` Functions

These functions return the input string in all uppercase or all lowercase letters, respectively. For example, consider the following script.

```
<?php
    $inquote = "Now Is The Time";
    $lower = strtolower($inquote);
    $upper = strtoupper($inquote);
    print ("upper=$upper lower=$lower");
?>
```

The above PHP script would output “upper=NOW IS THE TIME lower=now is the time”.

The `substr()` Function

This function enables a PHP script to extract a portion of the characters in a string variable. It has the general syntax shown in Figure 2.6.

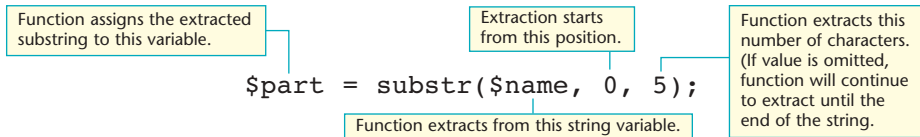


Figure 2.6 General Syntax of the `substr()` Function

The `substr()` function returns a substring from the string specified in the first argument. The function starts at a specific character position (the second argument) and extracts a specific number of characters (the third argument). If the third argument is omitted, then the `substr()` function will extract from the starting position (the second argument) until the end of the string.

It is important to note that the `substr()` function enumerates character positions starting with 0 (not 1), so, for example, in the string "Homer", the "H" would be position 0, the "o" would be position 1, the "m" position 2, and so on.

As an example, consider the following PHP script.

```
<?php
    $date = "12/25/2002";
    $month = substr($date, 0, 2);
    $day = substr($date, 3, 2);
    print ("Month=$month Day=$day");
?>
```

The above PHP segment would output "Month=12 Day=25".

As another example, consider the following use of the `substr()` function that does not include the third argument (and thus returns a substring from the starting position to the end of the search string).

```
<?php
    $date = "12/25/2002";
    $year = substr($date, 6);
    print ("Year=$year");
?>
```

The above script segment would output "Year=2002".

★ TIP Using a Negative Starting Value as a `substr()` Argument

If you use a negative value for the second argument in the `substr()` function, the function will determine the starting position counting from the right end of the string instead of the left. For example, the following use of `substr()` extracts a substring that starts four characters from the end of the string.

```
<?php
    $filename = "mydoc.html";
    $suffix = substr($filename, -4);
    print ("suffix=$suffix");
?>
```

The above script would output "suffix=html".

Creating HTML Input Forms

So far the PHP scripts we have developed have been limited in that they cannot receive any input data from the user. That is, before we used any variable we had to explicitly set an initial value inside the script. HTML forms provide elements such as text areas, check boxes, selection lists, and radio buttons to provide input data for your PHP scripts. (See an example of an HTML form in Figure 2.7.) Although HTML forms are not a part of the PHP language, their use is important for creating input for PHP scripts. Let's review the various HTML form elements and then see how to use them to receive data into PHP scripts.

Creating HTML Input Forms

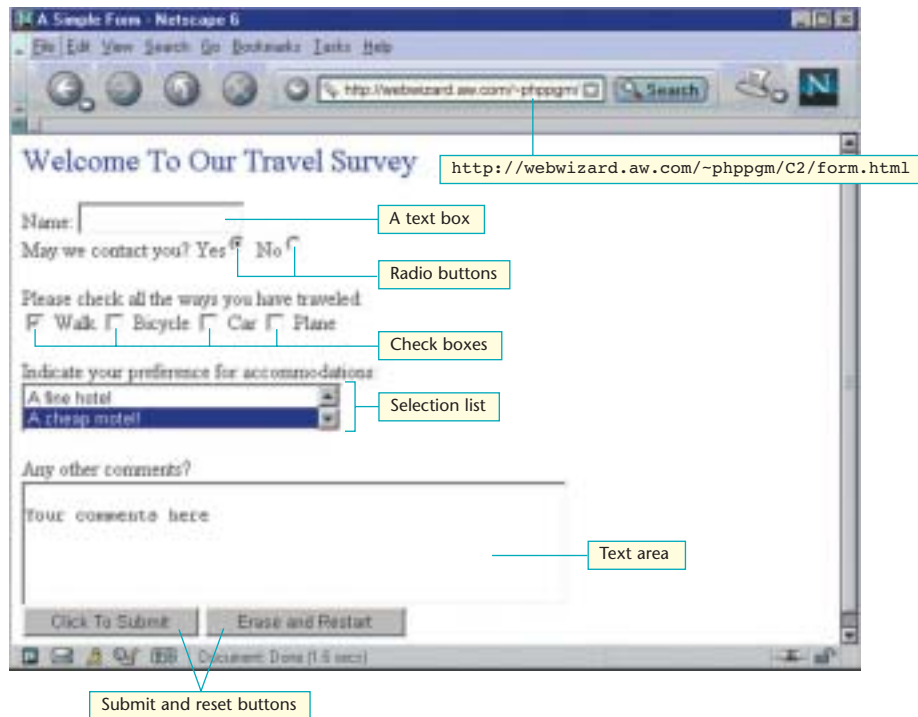


Figure 2.7 Example of a Simple HTML Form

Starting and Ending HTML Forms

You can create HTML forms by using the HTML `<form>` and `</form>` tags. Within these tags you place the various HTML form elements such as text areas, check boxes, and radio buttons. Figure 2.8 shows an example using the HTML `<form>` and `</form>` tags.

```

<form action="http://webwizard.aw.com/~phppgm/program.php"
      method="post">
    .
    .
    .
</form>
    
```

Figure 2.8 Format for Using the HTML `<form>` and `</form>` Tags

The `<form>` tag has two primary arguments:

1. **action.** This option specifies the web address or URL of the script to start when the form is submitted. When the example form from Figure 2.8 is submitted, a script at Web address `http://webwizard.aw.com/~phppgm/program.php` will be started and it will be able to receive any input sent from the form.
2. **method.** This argument is set to either `post` or `get`. It defines the argument format to use to send data to the PHP script. The `get` method appends the form arguments to the end of the Web address. The `post` method sends the data as part of the body of the HTML document. Since the `get` method can limit the amount of data you can send, we will use the `post` method exclusively.

Creating Form Buttons

Perhaps the most basic HTML form elements are form buttons, which enable the user to submit the form or erase all input and start again. When the user submits the form, it is sent to the location specified in the `action` argument of the `<form>` tag. HTML form buttons have the following format:

```

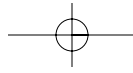
<input type="submit" value="Click To Submit">
<input type="reset" value="Erase and Restart">
    
```

Figure 2.9 Format for Creating HTML Form Submit and Reset Buttons

The HTML code in Figure 2.9 creates a submit button and a reset button. The submit button is labeled “Click To Submit” and the reset button is labeled “Erase and Restart.”

★ TIP **Another Argument for the Submit Button Form Element**

The submit button also has a `name` argument that is most commonly used when you have multiple submit buttons on a form and want to determine in the receiving script which button the user clicked.



To see a more complete example, consider the following script and its output in Figure 2.10. This script creates two buttons: a submit button and a reset button. When the form is submitted, it runs our initial PHP script from Chapter One, Figure 1.3 (saved at Web address <http://webwizard.aw.com/~phppgm/First.php>). The top portion of Figure 2.10 shows the output form from the HTML code. The bottom of Figure 2.10 shows the output from <http://webwizard.aw.com/~phppgm/First.php> when the form is submitted.

1. `<html>`
2. `<head> <title> A Simple Form </title> </head>`
3. `<body>`
4. `<form action="http://webwizard.aw.com/~phppgm/First.php" method="post" >`
5. Click submit to start our initial PHP program.
6. `
 <input type="submit" value="Click To Submit">`
7. `<input type="reset" value="Erase and Restart">`
8. `</form>`
9. `</body> </html>`

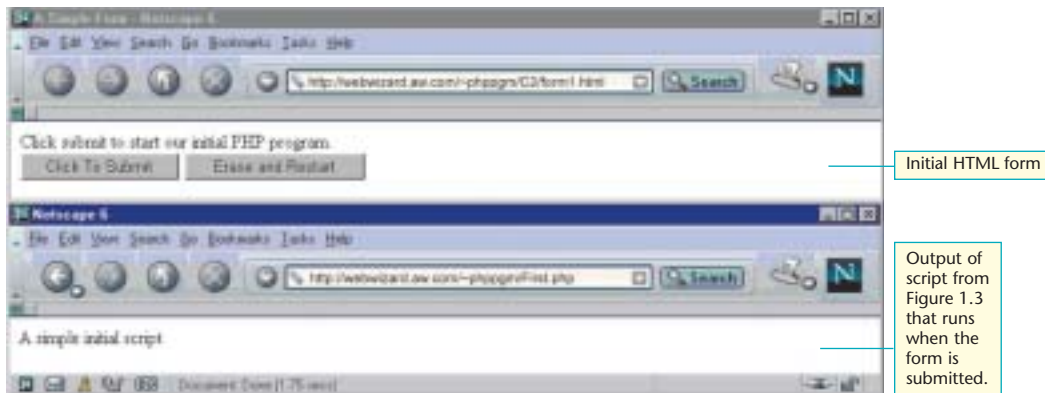


Figure 2.10 The Calling HTML Form (top) and the Output When the Form Is Submitted (bottom)

Let's briefly examine the key lines from this HTML document.

- ★ Lines 1–3 create the HTML tags needed to start an HTML document.
- ★ Line 4 starts the `<form>` tag and sets the `action` argument to start the script at <http://webwizard.aw.com/~phppgm/First.php>.
- ★ Lines 6–7 create the submit and reset buttons.
- ★ Lines 8–9 end the HTML form and HTML document.

Creating Text Input Boxes

Text input boxes create a form element for receiving a single line of text input. You can specify this form element within `<form>` and `</form>` tags as shown in Figure 2.11.

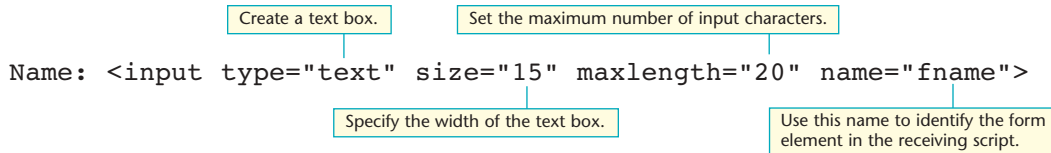


Figure 2.11 Format for Creating a Text Box HTML Form Element

The above HTML creates a text box 15 characters wide that will accept a maximum of 20 characters. It will set a variable named `fname` with value of whatever the end-user enters in the text box.

Creating Password Boxes

HTML supports creating text boxes as password areas instead of regular text boxes. Letters entered within a password box are viewed as asterisks (*) instead of the text being typed. To create a password box, set `type="password"` with the `input` form element tag. The other arguments of password boxes work much like those for text boxes. Figure 2.12 shows the HTML syntax used to create a password box within an HTML form. This example creates a password box that is 15 characters long and sets a variable called `pass1`.

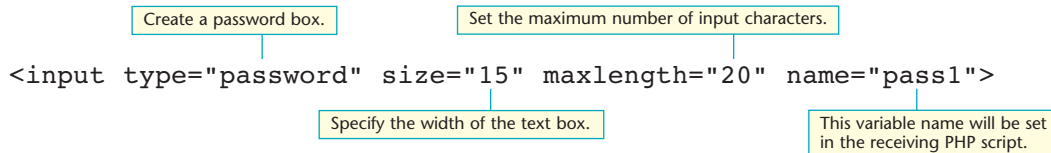


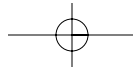
Figure 2.12 Format for Creating a Password Box HTML Form Element

★ WARNING Password Boxes Are Not Secure

The HTML password box is not a secure method for transmitting passwords. When the user submits the form, any data input is sent in clear text (nonencrypted) just like any other HTML form field. Someone with network access could, therefore, read the password being transferred. For this reason, most Web applications do not use this approach to receive and transmit passwords.

Creating Text Areas

Text areas are similar to text boxes except that with text areas you can create multiple column and multiple row text-input areas. The HTML code shown in Figure 2.13 creates a text area containing 4 rows and 50 columns. The words "Your comments here" are the default text and will appear automatically in the text area when the form starts. The variable name `Comments` will be available to the form-handling script. Its value will be set to whatever the user enters into the text area.



```

<textarea rows="4" cols="50" name="Comments">
Your comments here
</textarea>
    
```

Annotations:

- Set the number of rows. (points to rows="4")
- Set the number of columns. (points to cols="50")
- Any text here will appear as default text in text area. (points to Your comments here)
- Text areas have closing tags. (points to </textarea>)

Figure 2.13 Format for Creating a Text Area HTML Form Element

Creating Radio Buttons

Radio buttons are small circles that the user can select by clicking them with a mouse. Only one button within a group can be selected at any given time. As with other form elements, the `name` argument creates a name for the element. The `name` argument must be the same for all radio buttons that you want to operate together as a group. The `value` argument sets the variable value that will be available to the form-processing script. Figure 2.14 shows how to create radio buttons with HTML.

```

<input type="radio" name="contact" value="Yes" checked>
<input type="radio" name="contact" value="No">
    
```

Annotations:

- Create radio buttons. (points to <input type="radio" name="contact" value="Yes" checked>)
- Since both radio buttons have the same name, they will operate together. (points to name="contact" in both lines)
- This value will be sent to the form-processing program. (points to value="Yes")
- This item will appear checked when the user first views the form. (points to checked)

Figure 2.14 Format for Creating a Radio Button HTML Form Element

Creating Check Boxes

Check boxes are small boxes on a form that create a check mark when the user clicks them. The HTML lines shown in Figure 2.15 create four independent check boxes; that is, all four check box elements can be selected and each will set a value for a different variable name. For example, when the user clicks on the check box created in the second line, the variable `Bicycle` will have the value `Yes` in the receiving form-processing script.

```

<input type="checkbox" name="walk" value="Yes" checked> Walk
<input type="checkbox" name="Bicycle" value="Yes"> Bicycle
<input type="checkbox" name="Car" value="Yes"> Car
<input type="checkbox" name="Plane" value="Yes"> Plane
    
```

Annotations:

- Create check boxes. (points to <input type="checkbox" name="walk" value="Yes" checked>)
- Each check box sets a different variable name when selected. (points to name="Bicycle" in the second line)
- This item will appear checked when the user first views the form. (points to checked in the first line)
- This value will be sent to the form-processing program. (points to value="Yes" in the second line)

Figure 2.15 Format for Creating a Check Box HTML Form Element

Sometimes you might want to create a set of check boxes that use the same name argument. Unlike radio buttons, when the same name argument is used within a group of check boxes, multiple check box elements can still be selected. The value received by the form-processing script would be a comma-separated list of all items checked. For example, if the *second* and *third* items in the check boxes shown in Figure 2.16 were checked, the form-processing script would receive a variable called `travel` with two comma-separated values: "Bike, Horse". (Receiving comma-separated values from a check box name variable is covered in Chapter Five.)

This value will be sent to the form-processing program.

This item will appear checked when the user first views the form.

```
<input type="checkbox" name="travel" value="Car" checked> Car?
<input type="checkbox" name="travel" value="Bike" > Bicycle?
<input type="checkbox" name="travel" value="Horse"> Horse?
<input type="checkbox" name="travel" value="None"> None of the
above?
```

Create check boxes.

Since each check box element has the same name, multiple values can be set for the same variable name.

Figure 2.16 Format for Using Check Boxes with the Same name Argument

Creating Selection Lists

A selection list creates a box with a scrolling list of one or more items that the user can highlight and select. The entire list is enclosed in `<select>` and `</select>` tags. The `size` option defines how many lines will be displayed without scrolling. The `multiple` option allows more than one list item to be selected simultaneously. Within the `<select>` and `</select>` tags the HTML `<option>` tag defines each list option that will be displayed. The actual text displayed in the `<option>` tag is returned to the form-processing script through the variable specified in the name argument.

The HTML code shown in Figure 2.17 creates a selection list.

Specify the desired variable used in the receiving script.

Set the viewable window size.

This option allows users to select multiple items.

```
<select name="Accommodations" size="2" multiple>
<option> A fine otel </option>
<option selected> A cheap motel! </option>
<option> Just give me a sleeping back checked </option>
</select>
```

This text is displayed as an option and the entire text will be returned as the variable's value if selected.

Figure 2.17 Format for Creating a Selection List HTML Form Element

This HTML code creates four options formatted in a scrolling list. Only two of these options are displayed at the same time, and the user can select more than one option. Multiple selections are sent to the form-processing script as a comma-separated list. (Receiving comma-separated lists in a PHP script is covered in Chapter Five.)

Receiving Form Input into PHP Scripts

PHP makes it easy to receive input values from HTML forms. To receive input from HTML forms into your PHP script, use a PHP variable name that matches the variable defined in the form element's `name` argument. For example, assume the calling form uses a radio button with `name="contact"` as shown below.

```
<input type="radio" name="contact" value="Yes">
```

Then a form-handling PHP script can access the radio button's value by using a variable called `$contact`. In this case, if the user clicks the radio button, then when the form is submitted, `$contact` would have the value `Yes` in the receiving PHP script.

Consider the following full example of a PHP script receiving input from an HTML form that uses the following HTML form elements that create the variables `email` and `contact`.

```
Enter email address: <input type="text" size="16"
                    maxlength="20" name="email">
May we contact you?
<input type="radio" name="contact" value="Yes" checked> Yes
<input type="radio" name="contact" value="No"> No
```

The following script is placed into a file and stored at Web address `http://webwizard.aw.com/~phppgm/C2/PrRadio.php` (which matches the `action` argument of the calling form's `<form>` tag, not shown). The script will run when the form shown at the top of Figure 2.18 is submitted; the generated output appears at the bottom of Figure 2.18.

```
1. <html>
2. <head><title> Receiving Input </title> </head>
3. <body>
4. <font size="5">Thank You: Got Your Input.</font>
5. <?php
6.     print ("<br>Your email address is $email");
7.     print ("<br> Contact preference is $contact");
8. ?>
9. </body> </html>
```

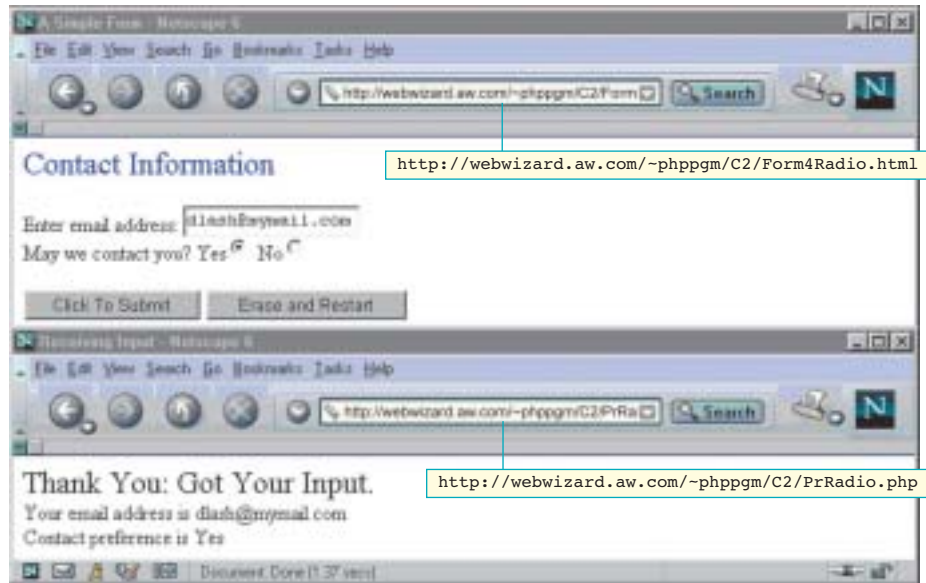


Figure 2.18 The HTML Input Form (top) and Output of a Script That Receives Input (bottom)

Here's a brief description of the script's key lines.

- ★ Line 6 outputs the value of `$email` set from the calling form in the text box form element.
- ★ Line 7 outputs the value of `$contact` set from the calling form in the radio button form element.

★ **SHORTCUT** Testing Input Values without HTML Forms

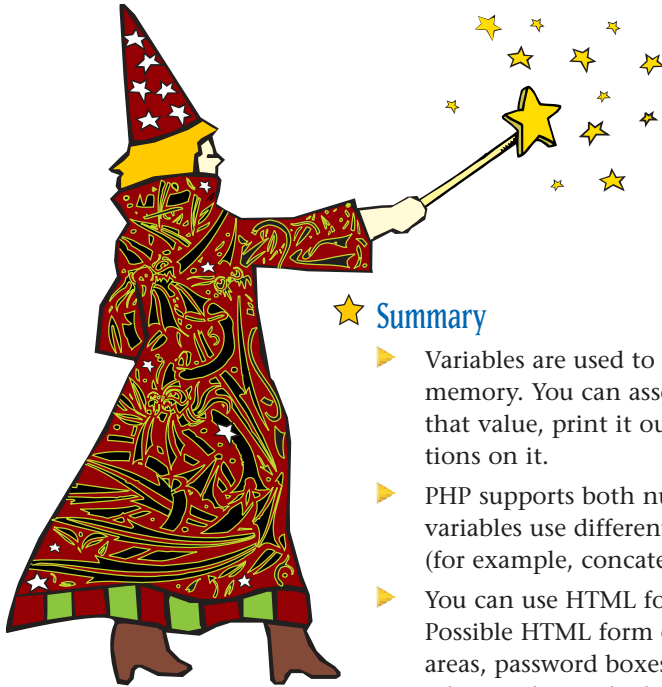
You can send test input data to PHP scripts without a front-end HTML form by specifying data values in the URL box of a browser. For example, a test value for the variable `$email` for the script output in Figure 2.18 can be sent by appending `?email=dlash@dlash.com` to the Web address in the browser as follows:

```
http://webwizard.aw.com/~phppgm/C2/PrRadio.php?email=dlash@dlash.com
```

You can also send a test value for `$email` and `$contact` by appending `?email=dlash@dlash.com&Contact=No` to the Web address in the browser as follows:

```
http://webwizard.aw.com/~phppgm/C2/PrRadio.php?email=dlash@dlash.com&contact=No
```

Sending input values directly from a URL (without a front-end form) can provide a powerful way to test your scripts. It also means that any user might send variable values to your scripts without using your front-end forms. This capability of sending data from a URL means your scripts have to check Web input data carefully.



★ Summary

- ▶ Variables are used to store and access data in computer memory. You can associate a value with a variable, change that value, print it out, and perform many different operations on it.
 - ▶ PHP supports both numeric and string variables. String variables use different methods for value manipulation (for example, concatenation) than numeric variables do.
 - ▶ You can use HTML forms to pass data to PHP scripts. Possible HTML form elements include text boxes, text areas, password boxes, check boxes, radio buttons, and selection lists, which allow users to input data for use in PHP scripts.
- ▶ PHP scripts can receive form element input values by using a PHP variable name that matches the one specified in the form element's name argument.

★ Online References

HTML tutorial that includes a form overview

<http://www.htmlgoodies.com/>

PHP tutorial and overview information

<http://hotwired.lycos.com/webmonkey/>

Article discussing how to use PHP to read form variables

http://www.devshed.com/Server_Side/PHP/Introduction/page3.html

Tutorial discussing strings in PHP

<http://www.zend.com/zend/tut/using-strings.php>

★ Review Questions

1. Which of the following are not valid PHP variable names?
`$1st_counter`, `$x1`, `$soccer`, `$TimeCounter`, `squared`
 Why are they invalid?
2. What are the operator precedence rules in PHP? What would be the values of `$x` and `$y` after executing the following statements?

```
<?php
    $x = 15 + 12 / 2 - 1;
    $y = (12 + 12) / 2 + 2;
?>
```

3. Name three numeric operators and three string manipulation functions.
4. What is the output of the following PHP code segment?

```
<?php
    $x = 12; $y = 4;
    $num = $x + 8 / $y;
    print ("Num is $num but x is $x");
?>
```

5. What is the output of the following PHP code segment?

```
<?php
    $x = 12; $y = 4; $z = 2;
    $num = $x * $z / $y;
    $x = $num * 2;
    print ("Num is $num but x is $x");
?>
```

6. What do the `trim()` and the `strtoupper()` functions do? What arguments do they use?
7. What is the output of the following PHP script?

```
<?php
    $file = "    data.txt";
    $short = trim($file);
    $part = substr($short, 0, 4);
    $part2 = substr($short, -3);
    print ("part=$part part2=$part2");
?>
```

8. What is the difference between the text box, text area, and password box HTML form elements?
9. Suppose you must write a PHP script to receive input using the following HTML form element. What PHP variable name would you use, and what would be its value if it were checked?
10. Suppose you must write a PHP script to receive input using the following HTML form element. What PHP variable name would you use? What would be the variable value if the user selected the third option?

```
<input type="checkbox" name="travel" value="Bike">
```

```
<select name="Favorites" size="2" multiple>
<option> Baseball </option>
<option selected> Wild Parties </option>
<option> Camping in the Wilderness </option>
<option> Mountain Biking </option>
</select>
```



★ Hands-On Exercises

- Write scripts that calculate the following equations. Print out your results as HTML documents.

- Calculate the volume of a cylinder using the following formula

$$V = \pi r^2 h$$

Use $r = 4$ and $h = 12$. Use 3.14 as the value of π .

- Calculate the average of four grades, where `grade1 = 100`, `grade2 = 75`, `grade3 = 98`, and `grade4 = 90`.
- Modify the script used in the example for Figure 2.3 to enable it to accept input from an HTML form. Create a front-end form that enables the user to set input values for ticket cost and building cost. After the script receives these values, output the total seats, total revenue, and total profit as shown in Figure 2.3.
 - Create a Web application that uses an HTML form to input the values of r and h used in Exercise 1a. Receive these values into a PHP script and output the results of the formula.
 - Create a form that includes a text box labeled `student_name` and four text boxes labeled `grade1`, `grade2`, `grade3`, and `grade4`. When the user enters his or her name and four grades, use a PHP script to return a new Web page with the user's name and average grade.
 - Create an HTML form that asks the user to enter a filename (from a text box on a form). Instruct the user to include a three-letter file suffix (for example, `.php`, `.txt`, or `.htm`). Create a receiving PHP script that outputs the file suffix (the last three characters of the filename) and the filename (all characters before the last four characters) that the user input. Remove any leading spaces that might be included.
 - Create a Web form that has the following form elements:
 - ★ A text box for receiving a first name
 - ★ Two radio buttons asking if the user purchases Internet access from an ISP (yes or no)
 - ★ A checklist (with the same variable name set for each item) that asks the user to check all the different ways he or she has used the Web:
 - ★ For research
 - ★ To purchase items electronically
 - ★ To send and receive e-mail
 - ★ To read news

Receive the above input into a PHP script and output all of the user's input. Also output the number of characters in his or her first name.